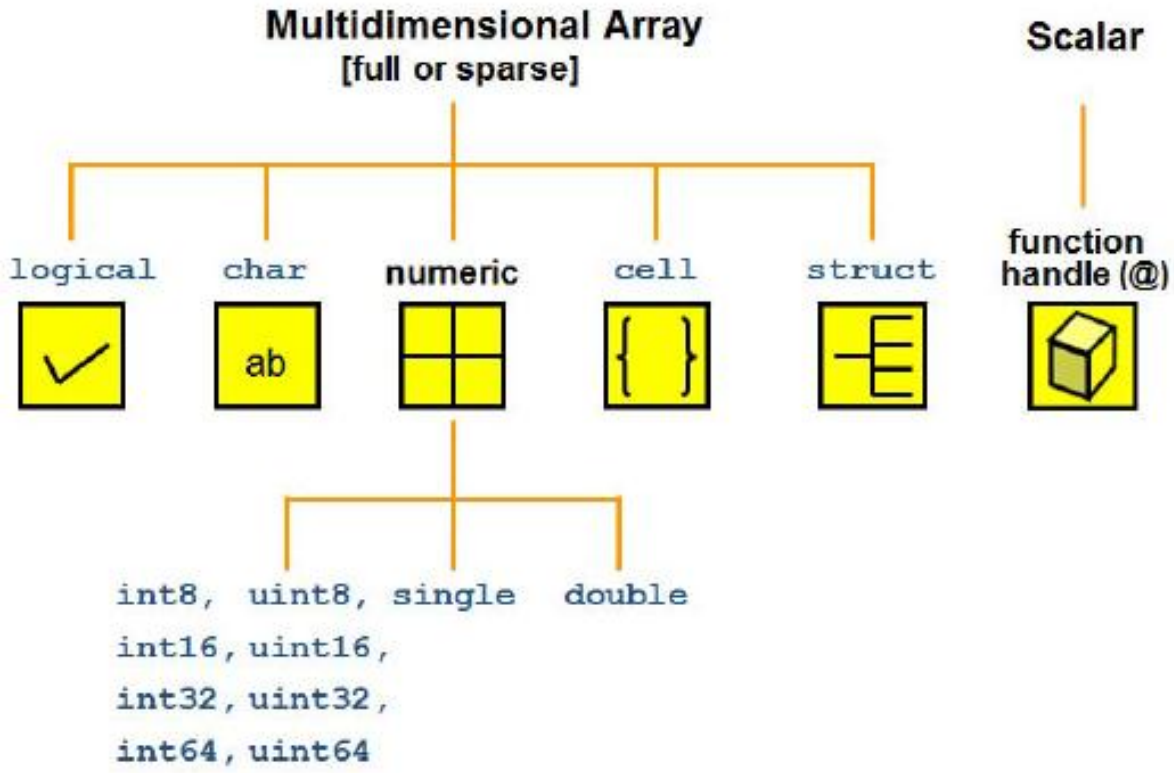
Mini Tutorial de MATLAB

Jorge De Los Santos

Tipos de datos



Operadores aritméticos, relacionales y lógicos

Operator	Description
+	Addition
-	Subtraction
.*	Multiplication
./	Right division
.\	Left division
+	Unary plus
-	Unary minus
:	Colon operator
.^	Power
.'	Transpose
'	Complex conjugate transpose
*	Matrix multiplication
/	Matrix right division
\	Matrix left division
^	Matrix power

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Operator	Description	Example
&	Returns 1 for every element location that is true (nonzero) in both arrays, and 0 for all other elements.	A & B = 01001
|	Returns 1 for every element location that is true (nonzero) in either one or the other, or both arrays, and 0 for all other elements.	A | B = 11101
~	Complements each element of the input array, A.	~A = 10010
xor	Returns 1 for every element location that is true (nonzero) in only one array, and 0 for all other elements.	xor(A,B) = 10100

Funciones / comandos básicos de MATLAB

Función	Descripción
clear	Limpia variables del workspace
clc	Limpia la ventana de comandos
whos	Muestra información acerca de las variables existentes
class	Muestra la clase de la variable pasada como argumento
which	Muestra la ubicación de una función
exit	Cierra MATLAB
quit	Cierra MATLAB
format	Configura el formato de salida
save	Guarda variables del workspace
load	Carga variables almacenadas en un archivo-MAT

Operaciones aritméticas básicas

```
>> 3+2

ans =

     5

>> 7/5

ans =

    1.4000
```

```
>> 4*8

ans =

    32

>> 2-5+2-1

ans =

    -2

>> 5^3

ans =

   125
```

Funciones matemáticas y constantes predefinidas

Funciones trigonométricas (El argumento de entrada debe ser un ángulo en radianes)

```
>> theta=pi/4

theta =

    0.7854

>> cos(theta)

ans =

    0.7071

>> sin(theta)

ans =

    0.7071

>> tan(theta)

ans =

    1.0000

>> sec(theta)

ans =
```

```
    1.4142
>> csc(theta)
ans =
    1.4142
>> cot(theta)
ans =
    1.0000
```

Funciones trigonométricas inversas (Los argumentos de salida están dados en radianes)

```
>> x=0.5;
>> acos(x)
ans =
    1.0472
>> asin(x)
ans =
    0.5236
>> atan(x)
ans =
    0.4636
```

Otras funciones matemáticas:

```
>> sqrt(5)
ans =
    2.2361
>> exp(3)
ans =
   20.0855
```

```
>> log(exp(1))
ans =
     1
>> log10(1000)
ans =
     3
```

Constantes predefinidas

```
>> pi
ans =
    3.1416
>> exp(1)
ans =
    2.7183
>> eps
ans =
   2.2204e-16
>> realmax
ans =
  1.7977e+308
>> realmin
ans =
  2.2251e-308
```

Crear variables

En MATLAB la asignación de variables se hace utilizando el signo “igual”. El nombre de una variable puede contener sólo caracteres alfanuméricos, pero no puede comenzar con un número.

```
>> a=3;

>> b=4;

>> c=5;

>> suma=a+b+c

suma =

    12

>> promedio=suma/3

promedio =

     4

>> sumaCuadrados=a^2+b^2+c^2

sumaCuadrados =

    50
```

Identificar tipos de datos

Para identificar los tipos de datos puede utilizarse el comando `whos` o bien la función `class`, por ejemplo:

```
>> txt='Hola Mundo';

>> vlog=true;

>> num=45;

>> entero=int8(12);

>> z=complex(2,3);

>> whos

  Name        Size            Bytes  Class      Attributes
```

```
  entero      1x1                 1  int8
  num         1x1                 8  double
  txt         1x10               20  char
  vlog        1x1                 1  logical
  z           1x1                16  double     complex
```

```
>> class(z)
ans =
double
>> class(vlog)
ans =
logical
>> class(entero)
ans =
int8
>> class(txt)
ans =
char
```

Guardar y cargar variables del workspace

Para guardar variables MATLAB proporciona la función `save`, véase el ejemplo siguiente:

```
>> p=3;
>> q=2.5;
>> save('variables.mat')
```

La instrucción anterior guarda en el archivo “variables.mat “ todas las variables existentes en el workspace. Es posible seleccionar las variables a guardar introduciendo un segundo argumento de entrada:

```
>> save('variables.mat','p');
```

Para cargar las variables guardadas en un archivo-MAT use la función load:

```
>> load('variables.mat');
```

Números complejos

Los números complejos en el entorno de MATLAB pertenecen la clase double, añadiéndose el atributo "complex" para diferenciarlos de los tipos double reales.

¿Cómo declarar un número complejo? Puede hacerse de las siguientes maneras:

```
>> 2+3i

ans =

   2.0000 + 3.0000i

>> 2+3j

ans =

   2.0000 + 3.0000i

>> complex(2,3)

ans =

   2.0000 + 3.0000i
```

Operaciones con números complejos.

Suma, resta, multiplicación y división.

```
>> z1=complex(4,7);

>> z2=complex(-3,1);

>> z1+z2

ans =

   1.0000 + 8.0000i

>> z1-z2
```

```
ans =

   7.0000 + 6.0000i

>> z1*z2

ans =

 -19.0000 -17.0000i

>> z1/z2

ans =

  -0.5000 - 2.5000i
```

Módulo, argumento y conjugado.

```
>> z=complex(1.5,2);

>> abs(z)

ans =

    2.5000

>> angle(z)

ans =

    0.9273

>> conj(z)

ans =

   1.5000 - 2.0000i
```

Vectores y matrices

En MATLAB el tipo de dato predilecto son las matrices (de ahí la derivación del nombre MAtrix LABoratory), todo en MATLAB es una matriz, incluso los escalares se consideran una matriz de dimensión unitaria.

¿Cómo crear un vector?

Para crear un vector deben introducirse entre corchetes los elementos que conforman el vector, separados estos por espacios o bien por “comas”, tal como se muestra enseguida:

```
>> v1=[3 0 -2 8 11 2 1]

v1 =

     3     0    -2     8    11     2     1

>> v2=[1,7,-2,4,9,3,-5]

v2 =

     1     7    -2     4     9     3    -5
```

¿Cómo crear una matriz?

Una matriz en MATLAB se introduce colocando las filas separadas con un punto y coma, a su vez, como en el caso de los vectores, los elementos de una misma fila se separan mediante espacios o comas, véanse los siguientes ejemplos:

```
>> A=[1 2 3;4 5 6;7 8 9]

A =

     1     2     3

     4     5     6

     7     8     9

>> B=[1,2,3;4,5,6;7,8,9]

B =

     1     2     3

     4     5     6

     7     8     9
```

Operaciones básicas con matrices

Por cuestiones de comodidad y para evitar conflictos al momento de realizar las operaciones utilizaremos dos matrices cuadradas de las mismas dimensiones (tomar en cuenta las definiciones básicas del álgebra lineal respecto al producto y suma de matrices). Sean **A** y **B** las matrices que enseguida se definen:

```
>> A=[-1 3;2 4]

A =
```

```
    -1     3
     2     4
>> B=[1 -4;5 2]
B =
     1    -4
     5     2
```

Una vez creadas las matrices anteriores podemos realizar algunas operaciones básicas como se muestra enseguida:

```
>> A+B
ans =
     0    -1
     7     6
>> A-B
ans =
    -2     7
    -3     2
>> B-A
ans =
     2    -7
     3    -2
>> A*B
ans =
    14    10
    22     0
>> A^2
ans =
     7     9
```

```
     6    22
>> B^2+A
ans =
   -20    -9
    17   -12
>> B^3-A^2
ans =
   -86    43
   -71  -114
```

Determinante e inversa de una matriz

Para calcular el determinante de una matriz utilizamos la función `det`, y para la inversa la función `inv`:

```
>> M=[1 -2 0;10 3 -5;1 4 2]
M =
     1    -2     0
    10     3    -5
     1     4     2
>> det(M)
ans =
   76.0000
>> inv(M)
ans =
    0.3421    0.0526    0.1316
   -0.3289    0.0263    0.0658
    0.4868   -0.0789    0.3026
```

Producto escalar y vectorial

Sean **u** y **v** los vectores definidos como sigue:

```
>> u=[-2 3 1]

u =

    -2     3     1

>> v=[1 1 2]

v =

     1     1     2
```

Podemos calcular el producto escalar mediante la función dot y el producto vectorial con cross:

```
>> dot(u,v)

ans =

     3

>> cross(u,v)

ans =

     5     5    -5
```

Indexado de matrices

Para acceder a un determinado elemento de una matriz se utiliza la siguiente notación:

```
>> A(i,j)
```

Donde i hace referencia a la i-ésima fila y j a la j-ésima columna en la cual se ubica el elemento. Por ejemplo, sea la matriz A definida por:

```
>> A=[1 -3 7;0 1 -1;2 8 5]

A =

     1    -3     7

     0     1    -1

     2     8     5
```

Si queremos visualizar el elemento que se ubica en la segunda fila y tercera columna, habremos de colocar la siguiente instrucción:

```
>> A(2,3)

ans =

    -1
```

Para visualizar toda la primera fila sería:

```
>> A(1,:)

ans =

     1    -3     7
```

Para mostrar todos los elementos de la segunda columna:

```
>> A(:,2)

ans =

    -3

     1

     8
```

Para remplazar un determinado elemento, basta con igualar la posición del elemento al nuevo valor, por ejemplo:

```
>> A(2,2)=10

A =

     1    -3     7

     0    10    -1

     2     8     5
```

Ahora, supongamos que se requiere que todos los elementos que sean negativos sean sustituidos por un valor de cero, para ello se utilizaría lo siguiente:

```
>> A(A<0)=0

A =

     1     0     7

     0    10     0

     2     8     5
```

Ficheros de comandos

Los ficheros de comandos, conocidos también como "**scripts**", son archivos de texto sin formato (ASCII) con la extensión característica de los archivos de MATLAB (*.m), se utilizan para almacenar una serie de comandos o instrucciones que se ejecutan sucesivamente y que habrán de realizar una tarea específica. Los scripts de MATLAB pueden editarse utilizando cualquier editor de texto sin formato (Bloc de Notas, Notepad++, Sublime Text, etc…), aunque es más recomendable utilizar el editor de MATLAB, puesto que proporciona herramientas que facilitan la corrección de errores, el control sobre la ejecución del código y la capacidad de autocompletado y sugerencias cuando se utilizan funciones nativas de MATLAB.

Para crear un nuevo script puede pulsar la combinación **Ctrl + N** (bajo SO Windows), o buscar en la interfaz de MATLAB la opción **New** y enseguida seleccionar **Script**; si prefiere hacerlo desde la ventana de comandos puede introducir el comando `edit` que le abrirá un nuevo script.

Para guardar un fichero de comandos utilice la opción **Save** de la barra de herramientas o bien mediante la combinación de teclas **Ctrl + S** en Windows. Debe tomarse en cuenta que al guardar un script se le proporcione un nombre que no entre en conflicto con las funciones nativas de MATLAB o las palabras reservadas del lenguaje. Algunas recomendaciones que deben seguirse para nombrar un script son:

- El nombre deberá contener sólo letras, números o guiones bajos.
- No deberá comenzar con un carácter diferente a una letra (Por ejemplo: `102metodo.m`, es un nombre inválido dado que comienza con un número).
- Evite utilizar nombres de funciones nativas de MATLAB o palabras reservadas del lenguaje que podrían ocasionar conflictos.

Entradas y salidas en el Command Window

La función `input`

La función `input` permite "pedir" un valor al usuario utilizando una cadena de caracteres como *prompt*, la sintaxis es muy sencilla:

```
var=input('Introduzca un valor: ');
```

En la variable **`var`** se guarda el valor que el usuario introduzca, los valores aceptados por la función input pueden ser de tipo numérico, cell arrays, e inclusive tipo char. Aunque para introducir cadenas de texto la

función `input` dispone de un modificador que hará que la entrada se evalúe como una variable tipo char o cadena de texto, la sintaxis para esto es la siguiente:

```
var=input('Introduzca una cadena de texto: ', 's');
```

La letra s entre comillas simples le indica a MATLAB que deberá evaluar la entrada como tipo string.

Salida sin formato: la función `disp`

La función `disp` muestra en pantalla el valor de una determinada variable que se pasa como argumento, por ejemplo:

```
>> a=3;

>> disp(a)

     3
```

Para el caso anterior se pasa como argumento la variable **`a`** que ha sido declarada previamente y simplemente se muestra el valor correspondiente a esta. Las variables a mostrar pueden ser de cualquier tipo, incluyendo cadenas de texto, matrices, cell arrays y estructuras, véanse los siguientes ejemplos:

```
>> disp(magic(3))

     8     1     6

     3     5     7

     4     9     2

>> disp({1,0,2,-2})

    [1]    [0]    [2]    [-2]

>> disp('Hola Mundo')

Hola Mundo
```

Con `disp` también es posible mostrar enlaces a un sitio web, utilizando la sintaxis HTML para un enlace dentro de la función `disp`, por ejemplo:

```
>> disp('<a href="http://matlab-typ.blogspot.mx">MATLAB TYP</a>')
```

Salida con formato: la función `fprintf`

Con `fprintf` es posible dar formato a la salida que se quiere imprimir en pantalla, por ejemplo, es posible especificar el número de decimales que se mostrarán o bien si se quiere mostrar como un entero o quizá como una cadena de texto. La sintaxis de la función fprintf es como sigue:

```
fprintf('Especificaciones de formato',a1,...,an);
```

Donde las especificaciones de formato incluyen uno o más de los identificadores de un mismo tipo o combinados que se muestran en la siguiente tabla:

IDENTIFICADOR	FORMATO DE SALIDA
`%d`	Notación decimal
`%f`	Tipo coma flotante
`%g`	Tipo coma flotante compacta.
`%u`	Tipo entero sin signo
`%e`	Tipo coma flotante, notación exponencial
`%s`	Tipo char, cadena de texto
`%c`	Tipo char, carácter a carácter.

Véase el siguiente ejemplo:

```
>> fprintf('%d',pi);
3.141593e+00>>
```

Observe que se imprime el valor de π en este caso, pero el *prompt* de la ventana de comandos queda situado justo después del valor de salida en la misma línea, para evitar lo anterior puede utilizar la secuencia de escape `\n` después del valor a imprimir, lo cual le indica a MATLAB que debe comenzar en una nueva línea. Modificamos y vemos el resultado que produce:

```
>> fprintf('%d\n',pi);
3.141593e+00
```

Ahora observe lo que se imprime utilizando otros identificadores:

```
>> fprintf('%f\n',pi);

3.141593

>> fprintf('%g\n',pi);

3.14159

>> fprintf('%e\n',pi);

3.141593e+00

>> fprintf('%u\n',pi);

3.141593e+00
```

Para las salidas de coma flotante puede especificar el número de decimales que tendrá la salida, por ejemplo si desea mostrar solamente dos decimales del número π:

```
>> fprintf('%.2f\n',pi);

3.14
```

Estructuras de control y bucles

Sentencia `if-elseif-else`

La sentencia if se utiliza como bifurcación simple por sí sola, es decir, en aquellas situaciones en las cuales se requiera evaluar solamente una condición, por ejemplo, suponga que tiene dos números a y b y necesita comprobar si son iguales y ejecutar una acción, para ello bastaría con una sentencia `if` simple:

```
if a==b
    disp('a es igual a b');
end
```

A diferencia del caso anterior hay situaciones que requieren la ejecución de una acción cuando la condición se cumpla y de otra en caso contrario, entonces puede utilizarse una bifurcación doble formada por las sentencias `if-else`. Retomando el ejemplo para la bifurcación if simple, podríamos modificarlo de tal manera que envíe también un mensaje (ejecute una acción) para cuando la condición no se cumple:

```
if a==b
    disp('a es igual a b');
else
    disp('a es diferente de b');
end
```

Ahora imagine que para los ejemplos anteriores se necesita especificar si a=b, si a>b o bien si a<b, lo cual implicaría tener una sentencia de selección múltiple `if-elseif-else` que permite escoger entre varias opciones, evaluándose en orden descendente, por ejemplo refiérase a la siguiente estructura:

```
if cond1
    % Instrucciones
elseif cond2
    % Instrucciones
elseif cond3
    % Instrucciones
    .
    .
    .
elseif condN
    % Instrucciones
else
    % Instrucciones
end
```

MATLAB evalúa primeramente la condición 1 contenida en la sentencia if (cond1) y en el caso de no cumplirse evalúa la siguiente condición de forma sucesiva (cond2, cond3, …); finalmente y en el caso de que ninguna de las opciones evaluadas se cumpla, se ejecuta la instrucción contenida en la sentencia else. A continuación se muestra el ejemplo de una bifurcación múltiple para la situación descrita al principio:

```
if a==b
    disp('a es igual que b');
elseif a>b
    disp('a es mayor que b');
elseif a<b
    disp('a es menor que b');
end
```

Sentencia `switch`

La sentencia `switch` es una bifurcación múltiple que permite seleccionar entre varias opciones o casos la acción a ejecutar. La sintaxis general es:

```
switch var
   case opc1
      % Instrucciones
   case opc2
      % Instrucciones
    .
    .
    .
   otherwise
      % Intrucciones
end
```

Siendo **`var`** la variable que servirá como criterio de selección. Después de la palabra reservada `case`, se coloca el valor de **`var`** para el cual se ejecutarán esas instrucciones, y en `otherwise` se insertan las instrucciones que MATLAB deberá ejecutar por defecto en caso de no cumplirse ninguno de los casos especificados.

Enseguida se muestran dos ejemplos correspondientes a la sentencia de selección switch:

```
X=input('Inserte 0 o 1: ');
switch X
    case 0
        disp('Insertó cero');
    case 1
        disp('Insertó uno');
    otherwise
        warning('Valor incorrecto, verifique');
end
```

```
letra=input('Inserte una letra: ','s');
switch letra
    case {'a','e','i','o','u'}
        disp('Es una vocal');
    otherwise
        disp('Es una consonante');
end
```

Bucle `for`

La sintaxis general de un bucle `for` se muestra enseguida:

```
for i=inicio:incremento:fin
    % Instrucciones...
end
```

El valor **`inicio`** es a partir del cual se ejecutará el ciclo, el **`incremento`** es la cantidad que varía en cada paso de ejecución, y el valor de **`final`** establece el último valor que tomará el ciclo.

El siguiente código muestra un ciclo `for` muy básico, el cual simplemente muestra en consola el valor actual adquirido por la variable.

```
for i=1:10
    fprintf('Valor actual: %g \n',i);
end
```

Cuando no se especifica el incremento, como el caso anterior, MATLAB asume que es unitario.

Es posible utilizar ciclos `for` anidados, por ejemplo para cuando se requiere recorrer una matriz en sus dos dimensiones y ejecutar operaciones elemento por elemento. Véase el siguiente ejemplo:

```
A=round(rand(5)*10);
for i=1:5
    for j=1:5
        disp(A(i,j));
    end
end
```

Bucle `while`

El bucle `while` se utiliza, por lo general, cuando no se tiene un rango definido sobre el cual se realice la ejecución del ciclo o bien cuando la terminación del mismo viene dada por una condición. La sintaxis más común es:

```
while cond
    % Instrucciones
    % ...
    % ...
    % ...
end
```

Donde **cond** es la condición que determina la finalización de ejecución.

Enseguida se muestra un ejemplo muy básico que muestra en pantalla el valor de una variable utilizada como referencia:

```
k=1;
while k<10
    disp(k);
    k=k+1;
end
```

Lo anterior muestra en consola el valor de **k** mientras esta sea menor a 10, es decir muestra todos los valores enteros en el intervalo [1 9], es importante notar que la variable **k** debe incrementarse en cada ciclo para que en un momento determinado la condición de finalización se cumpla, de lo contrario se convertiría en un bucle infinito.

Ahora, veamos un ejemplo más práctico. La aproximación de una raíz cuadrada por el método babilónico implica realizar n iteraciones mediante la siguiente expresión:

$$r_n(x) = \frac{1}{2}\left(\frac{x}{r_{n-1}} + r_{n-1}\right)$$

Donde **x** es el número del cual se calcula la raíz cuadrada. A continuación se muestra el código implementado en MATLAB utilizando un bucle `while`:

```
x=input('Introduzca un número positivo: ');
r=x;
ra=0;
while ra~=r
    ra=r;
    r=(1/2)*(x/r+r);
end
fprintf('\nRaíz cuadrada de %g = %g\n\n',x,r);
```

Como se observa, en la variable **ra** se guarda la raíz aproximada calculada en una iteración anterior, de manera que esta sirva como comparación respecto a la nueva raíz calculada, el bucle termina cuando la diferencia entre el valor actual y el anterior es inferior a la tolerancia numérica (`eps`) soportada por MATLAB y por ende pasan a considerarse como valores iguales.

Gráficas en 2D

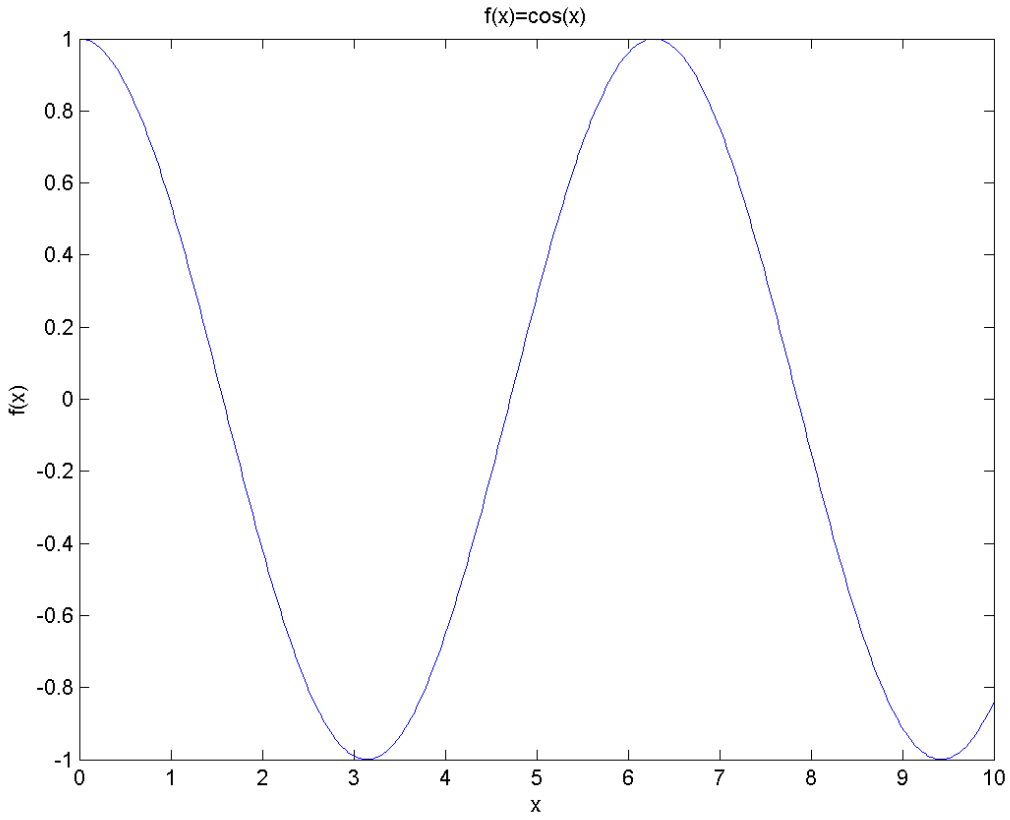
Gráficas en coordenadas rectangulares

Para trazar una gráfica en coordenadas rectangulares la función más utilizada es `plot` cuya sintaxis es:

```
>> plot(x,y);
```

Donde x e y son vectores del mismo tamaño que contienen información acerca de las coordenadas correspondientes. Véase el siguiente ejemplo en donde se grafica la función coseno:

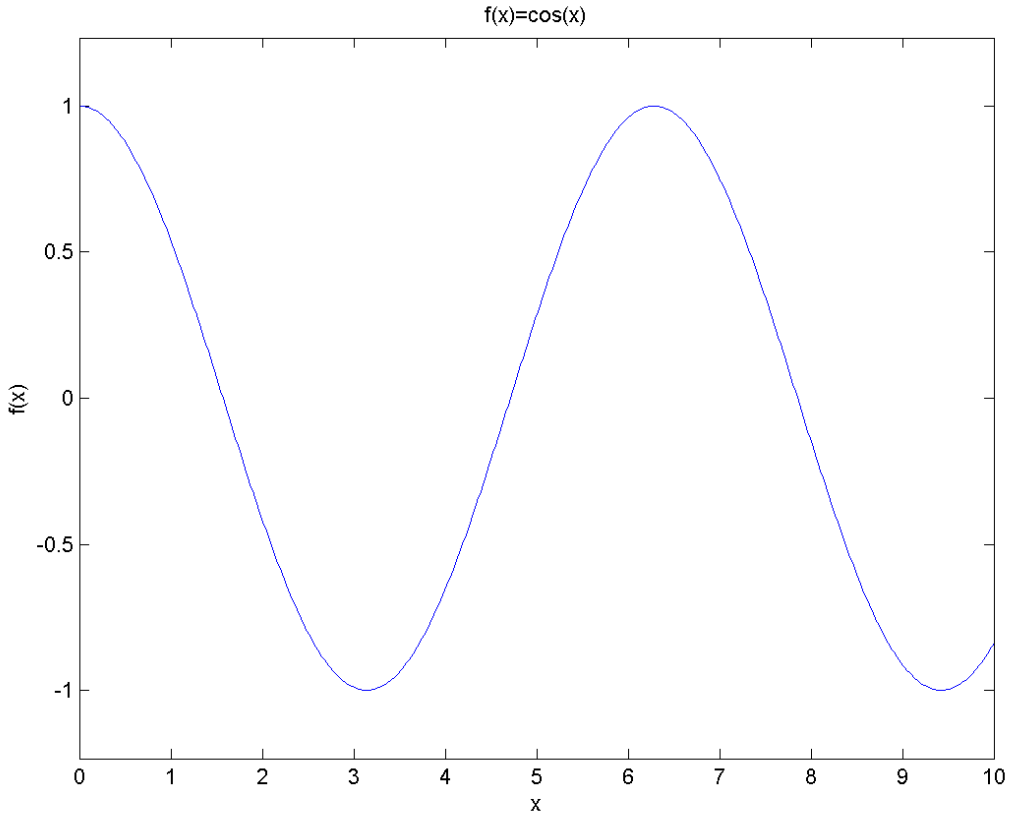
```
x=linspace(0,10,500);
y=cos(x);
plot(x,y);
xlabel('x');
ylabel('f(x)');
title('f(x)=cos(x)');
```



Existe una forma más simplificada para trazar la misma gráfica anterior, para ello se utiliza la función ezplot como se indica enseguida:

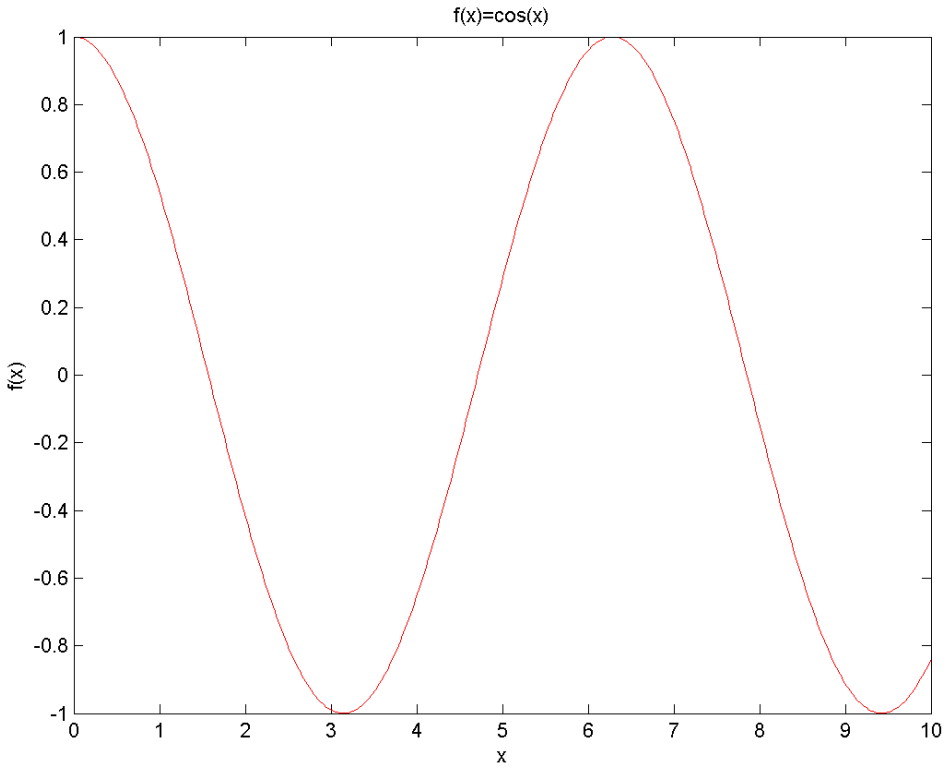
```
ezplot('cos(x)',[0 10]);
```

```
xlabel('x');
ylabel('f(x)');
title('f(x)=cos(x)');
```



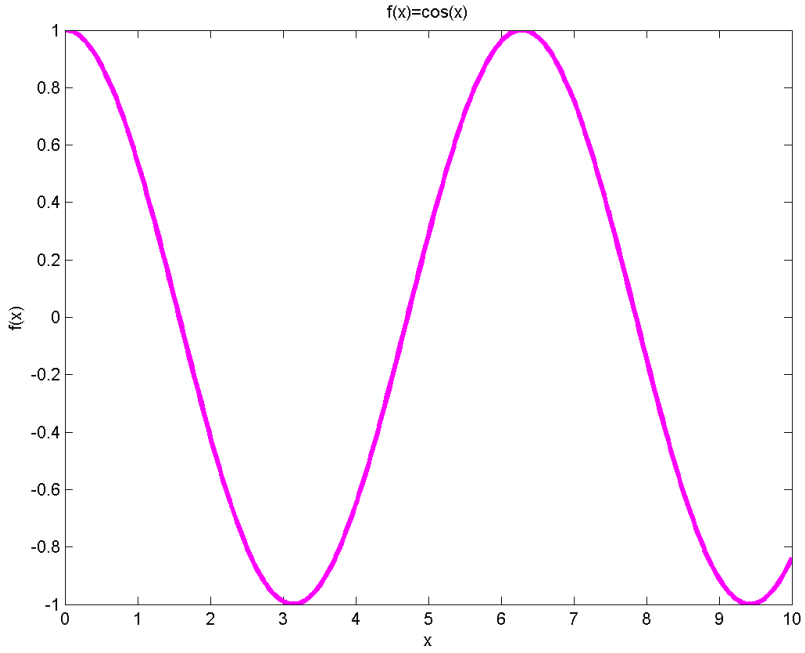
Modificando el color de línea.

```
x=linspace(0,10,500);
y=cos(x);
plot(x,y,'r'); % Línea en color rojo
xlabel('x');
ylabel('f(x)');
title('f(x)=cos(x)');
```



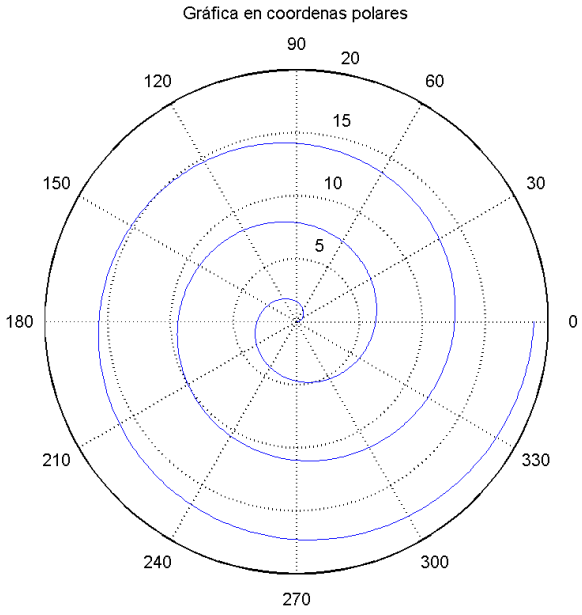
Modificando el grosor de línea

```
x=linspace(0,10,500);
y=cos(x);
plot(x,y,'m','linewidth',3);
xlabel('x');
ylabel('f(x)');
title('f(x)=cos(x)');
```



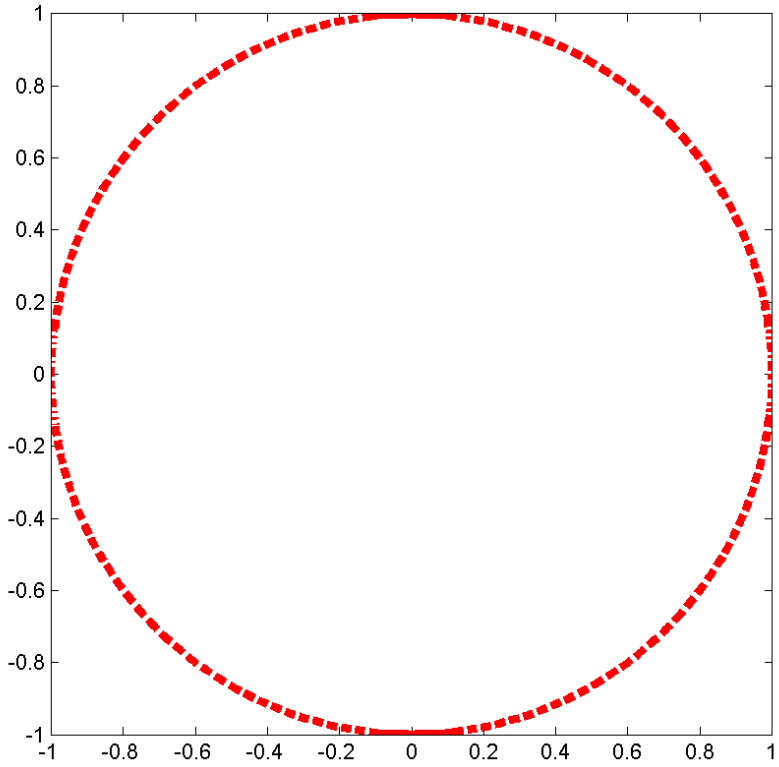
Gráficas en coordenadas polares.

```
theta=0:pi/180:6*pi;
r=theta;
polar(theta,r);
title('Gráfica en coordenas polares');
```



Gráfica de ecuaciones paramétricas

```
t=0:pi/180:2*pi;
x=cos(t);
y=sin(t);
plot(x,y,'r--','linewidth',4);
axis square;
```



Gráficas en 3D

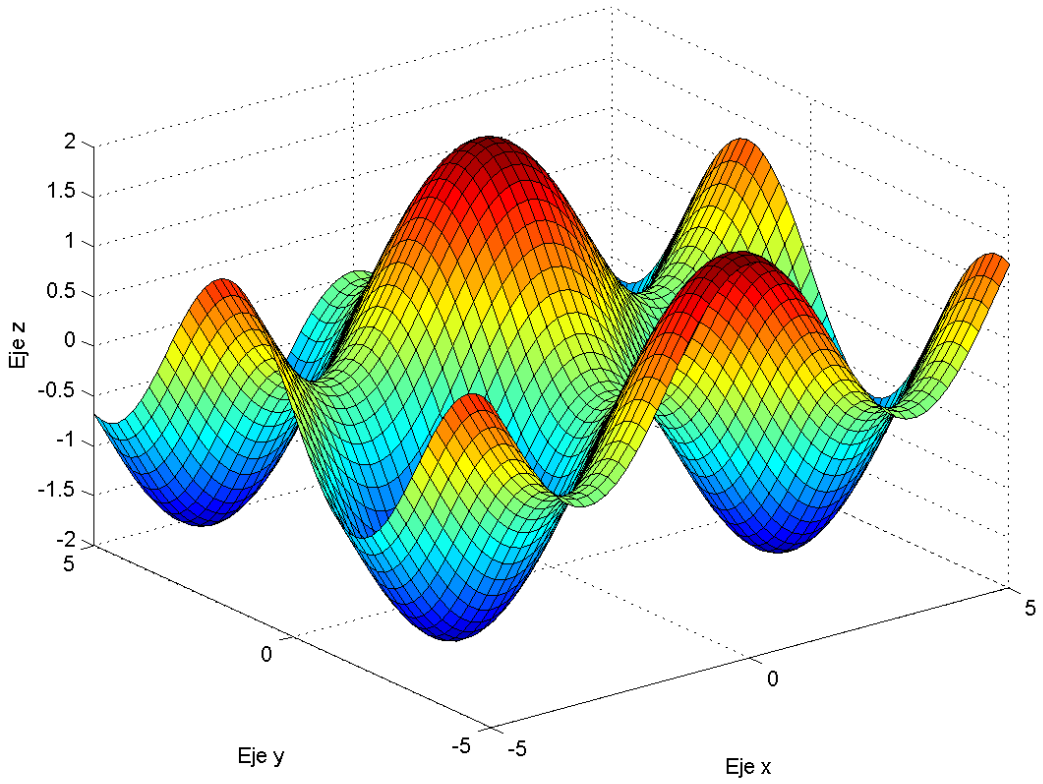
Gráfica de una superficie

Enseguida se muestra el ejemplo de la superficie definida por la función:

$$z(x, y) = \cos(x) + \sin(y)$$

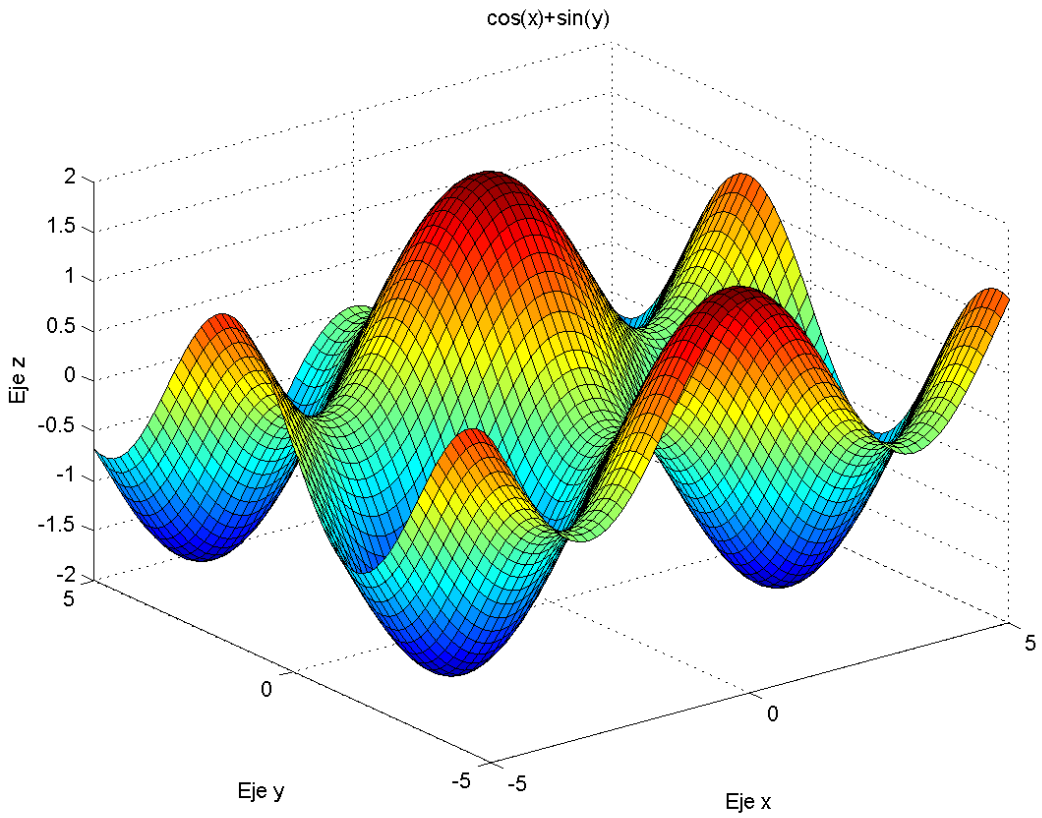
En el intervalo [-5 5] para ambas variables independientes.

```
[x,y]=meshgrid(-5:0.2:5);
z=cos(x)+sin(y);
surf(x,y,z);
xlabel('Eje x');
ylabel('Eje y');
zlabel('Eje z');
```



Puede utilizar la función simplificada `ezsurf` para trazar la misma superficie, véase el ejemplo:

```
ezsurf('cos(x)+sin(y)',[-5 5 -5 5]);
xlabel('Eje x');
ylabel('Eje y');
zlabel('Eje z');
```

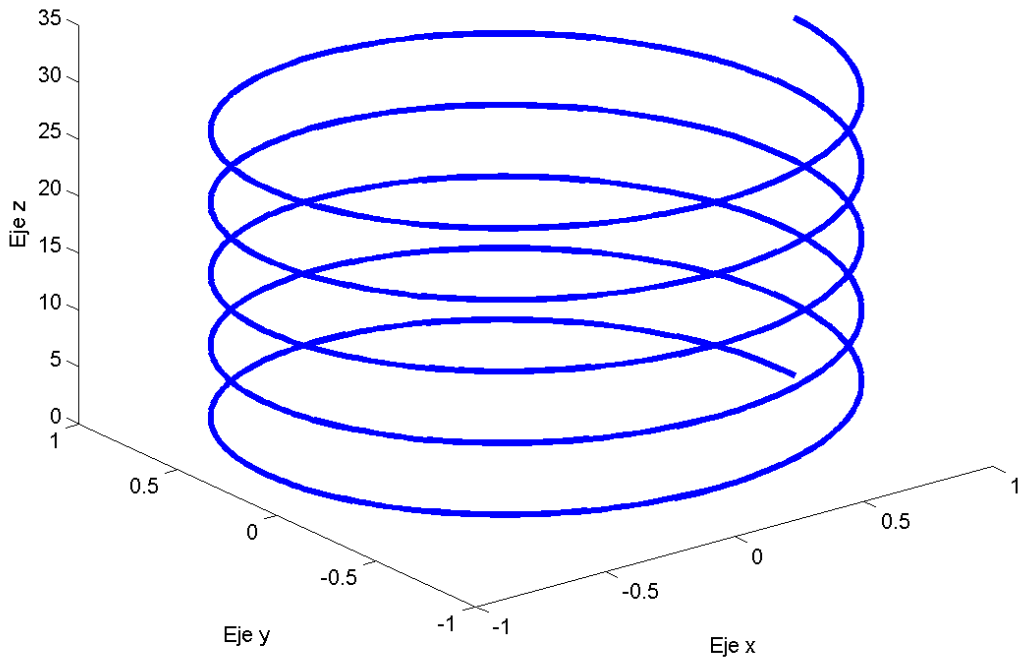


Gráfica de una curva paramétrica

A continuación se muestra la gráfica de una función cuyas ecuaciones paramétricas son:

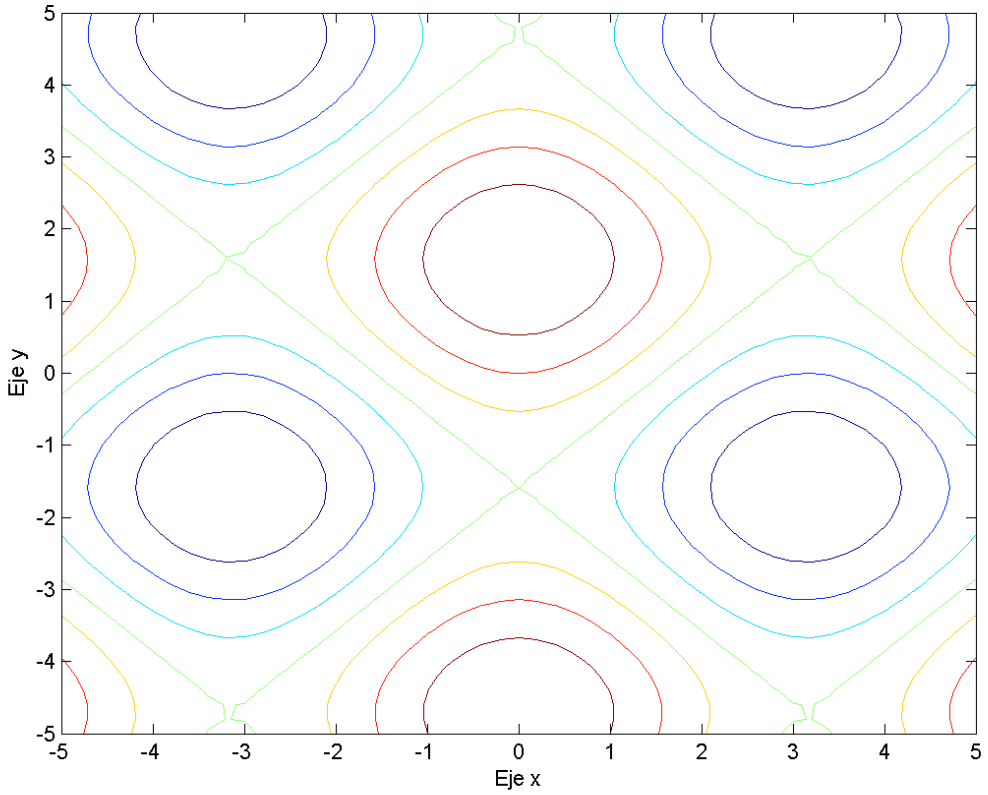
$$\begin{cases} x(t) = \cos(t) \\ y(t) = \sin(t) \\ z(t) = t \end{cases}$$

```
t=linspace(0,10*pi,1000);
x=cos(t);
y=sin(t);
z=t;
plot3(x,y,z,'linewidth',3);
xlabel('Eje x');
ylabel('Eje y');
zlabel('Eje z');
```



Gráfica de superficies de nivel

```
[x,y]=meshgrid(-5:0.2:5);
z=cos(x)+sin(y);
contour(x,y,z);
xlabel('Eje x');
ylabel('Eje y');
zlabel('Eje z');
```



Gráfica de una esfera

```
[x,y,z]=sphere;
surf(x,y,z);
daspect([1 1 1]);
```